EMI REGISTRY MANUAL

EMIR Product Team

Document Version:	1.2.0
Component Version:	1.2.0
Date:	11 07 2012

This work is co-funded by the EC EMI project under the FP7 Collaborative Projects Grant Agreement Nr. INFSO-RI-261611.

Contents

1 Overview

EMI Service Registry is a Service Endpoint Registry conceived during the EMI project. Its main goal is to discover all the Service Endpoints that exist. It consists of a collection of services that enables storing service records in a federated manner. Each of the record is a Service Endpoint Record (SER) complying with the OGF's GLUE 2.0 standard. The deployment of EMIR (which implies building an EMIR network over WAN) is bipartite: 1) Building a rooted hierarchy with a single EMIR server aggregating all the information within a federation 2) Sharing the information at the root level among peered EMIR servers (using P2P), thus enabling intra-federation discovery.

2 Features

The EMIR provides following features:

- the service endpoint record registration includes the management of the services' endpoint information.
- Powerful data back-end based on MongoDB
- Schema-free information model based on JSON (using GLUE2 entity names for specific attributes)
- REST-ful API to browse the service registrations
- Security
  - PKI governed authentication
  - Policy based authorisation

For more information about EMIR, visit EMI's TWiki[https://twiki.cern.ch/twiki/bin/view/-EMI/EMIRegistry].

3 Installation

In order to install EMIR, it is required to install the SUN or OpenJDK Java 6 (JRE or SDK). If not installed on the target system, it can be downloaded from http://java.oracle.com

- Linux based operating system
- MongoDB

EMIR is distributed in the following formats:

- Platform independent format, provided in "tar.gz" format
- RPM package, suitable SL5/SL6 and other Fedora based Linux derivatives (RedHat, CentOS etc. . .)
- Debian package

IMPORTANT NOTE ON PATHS
The location of the installation and configuration files differ depending on the type of bundle (see the above section).
If RPM bundle is being installed, the following paths will be used:

```
CONF=/etc/emi/emir
BIN=/usr/sbin
LOG=/var/log/emi/emir
```

The platform independent binary places all the files under single directory. The contents will be:

```
CONF=INST/conf/
BIN=INST/bin/
LOG=INST/logs/
```

The above variables (CONF, BIN and LOG) will be used throughout the rest of this manual.

3.1 Installation using the RPM bundle (RedHat Distributions)

Download EMIR Server's RPM distribution from the EMI's emisoft and install it using the rpm or yum command.

Example

```
yum install -y emi-emir
```

3.2 Installation using the Debian bundle (Centos/Debian Distributions)

Download EMIR DEB distribution from the EMI's emisoft and install it using the apt-get command.

3.3 Database Installation

EMIR server uses MongoDB database as a backbone to store and indexe SER collections. The database dependency will automatically be fetched from the **emisoft** repository, while installing

the EMIR Server. Otherwise it should be installed and configured before installing the EMIR. The installation and configuration instructions to setup the MongoDB database can be found on MongoDB's Web site.

3.4 Installation from the self-contained archive (tar.gz)

In order to generate, build and install the self contained binary it is required to follow the steps written below:

CREATING THE BUNDLE

1. check out the source code from **git://github.com/eu-emi/emiregistry.git**
2. go to **SOURCE_ROOT/emir-dist** directory
3. run **mvn assembly:assembly -DskipTests**

The archive can be found inside the **SOURCE_ROOT/target/emir-distribution-x.y.z-a-all.(tar.gz/zip)**, that contains all the necessary files for installation thus no special actions will be required except extraction to the target folder.

4 Configuration

The EMIR server comes with a well documented configuration file (CONF/emir.config), containing a number of options to setup registry hierarchy, p2p, security, http server, and database. The settings in the configuration file are pre-defined to start-up the server in a non-production environment, however the administrator needs to review before deploying on the production Grid environments.

4.1 General Configuration

The server configuration options in the CONF/emir.config are:

- Server address (plain or SSL)
- Settings of the type of the registry node, i.e. whether the current EMIR server instance is a child of some other (a parent) EMIR server node or a top/global registry in a hierarchy.

Property name	Type	Default value	Description
Server general settings			

Property name	Type	Default value	Description
`emir.address`	string	`http://localhost:9126`	The address/URL of the EMIR server on which it receives registration and query requests. It should either start with http or https (SSL/TLS) mode, if "https" mode is selected the Authentication and Authorisation properties must be properly configured
`emir.anonymousPort`	Unsigned Integer	`9127`	The anonymous http port number. Setting the property will start an additional *http* server (without SSL/TLS) only if the above server address is *https* (with SSL/TLS). It will provide *anonymous* access to the query interface (i.e. /services REST Web Service).

4.2 PKI Trust Settings Configuration

EMIR endorses Public Key Infrastructure (PKI) trust settings to validate certificates using EMI's caNL (JAVA version). The validation is performed when a connection with a remote peer is initiated over the network, using the SSL (or TLS) protocol, i.e. `emir.address` value has *https* scheme.

Certificates validation is primarily configured using a set of initially trusted certificates of so called Certificate Authorities (CAs). Those trusted certificates are also known as *trust anchors* and their collection is called as a *truststore*.

The validation mechanism except the *trust anchors* can use additional input for checking if a certificate being checked was not revoked and if its subject is in a permitted namesapce.

EMIR allows different types of truststores. All of them are configured using a set of specific properties in *CONF/emir.config* file.

4.2.1 OpenSSL Truststore

It allows using a directory with CA certificates stored in PEM format, with precisely defined names: Certificate Authorities (CA), Certificate Revocation List (CRL), signing policy and

namespaces files are named as <hash>.0, <hash>.r0, <hash>.signing_policy and <hash>.namespaces respectively. Hash is the old hash of the trusted CA certificate subject name - in OpenSSL version newer than 1.0.0 use -suject_hash_old switch to generate it. If multiple certificates have the same hash then the default zero number must be incremented. It is suggested when a common truststore with EMI (and Globus) middlewares is needed.

4.2.2 Directory Truststore

It allows to use a list of wildcard expressions, concrete paths of files, or URLs to remote files as a set of trusted CAs and CRLs. The truststore is configured as a directory containing all the trusted certificates (or with a specified extension). The directory with stored IGTF trust anchors can be set as a EMIR truststore for instance.

4.2.3 Java Keystore (JKS) Truststore

A single repository (or a binary file) of X.509 public key certificates with (optionally) accompanying private key certificates. The Java JDK already bundles keytool utility - a certificate manage utility to create JKS truststores.

4.2.4 PKCS#12 Truststore

Similar to JKS trustore, single binary file can be used to store X.509 public with (optionally) accompanying private key certificates. The **OpenSSL pkcs12** command can be used to parse, read, and create these files; the extension for PKCS#12 files is ".p12".

Property name	**Type**	**Default value / mandatory**	**Description**
`emir.security.truststore.allowProxy`	[ALLOW, DENY]	`ALLOW`	controls whether proxy certificates are supported
`emir.security.truststore.type`	[keystore, openssl, directory]	*mandatory to be set*	The truststore type.
`emir.security.truststore.reloadInterval`	integer number	`600`	how often the truststore should be reloaded, in seconds.
		--- Directory ---	
`emir.security.truststore.directory.connectionTimeout`	integer number	`15`	Connection timeout for fetching the remote CA certificates in seconds.

Property name	Type	Default value / mandatory	Description
emir.security.truststore.directoryDiskCacheDir	filesystem path		Directory where CA certificates should be cached, after downloading them from a remote source. Can be left undefined if no disk cache should be used. Note that directory should be secured, i.e. normal users should not be allowed to write to it.
emir.security.truststore.directoryEncoding	[PEM, DER]	PEM	For directory truststore controls whether certificates are encoded in PEM or DER.
emir.security.truststore.directoryLocations.*	list of properties with a common prefix		List of CA certificates locations. Can contain URLs, local files and wildcard expressions.
--- *Keystore* ---			
emir.security.truststore.keystoreFormat	string		The keystore type (jks, pkcs12) in case of truststore of keystore type.
emir.security.truststore.keystorePassword	string		The password of the keystore type truststore.
emir.security.truststore.keystorePath	string		The keystore path in case of truststore of keystore type.
--- *Openssl* ---			
emir.security.truststore.opensslNamespaceMode	[GLOBUS_EUGRIDPMA_REQUIRE, EUGRIDPMA_GLOBUS_REQUIRE, GLOBUS_REQUIRE, EUGRIDPMA_REQUIRE, EUGRIDPMA_AND_GLOBUS, EUGRIDPMA_AND_GLOBUS_REQUIRE, IGNORE]	GLOBUS_EUGRIDPMA_REQUIRE	In case of openssl truststore, controls which (and in which order) namespace checking rules should be applied

Property name	Type	Default value / mandatory	Description
emir.security.trustStore.opensslPath	filesystem path	/etc/grid-security/certificates	directory to be used for opeenssl truststore
		--- Revocation ---	
emir.security.trustStore.crlConnectionTimeout	integer number	15	Connection timeout for fetching the remote CRLs in seconds (not used for Openssl truststores).
emir.security.trustStore.crlDiskCachePath	filesystem path	-	Directory where CRLs should be cached, after downloading them from remote source. Can be left undefined if no disk cache should be used. Note that directory should be secured, i.e. normal users should not be allowed to write to it. Not used for Openssl truststores.
emir.security.trustStore.crlLocations.*	list of properties with a common prefix	-	List of CRLs locations. Can contain URLs, local files and wildcard expressions. Not used for Openssl truststores.
emir.security.trustStore.crlMode	[REQUIRE, IF_VALID, IGNORE]	IF_VALID	General CRL handling mode. The IF_VALID setting turns on CRL checking only in case the CRL is present.
emir.security.trustStore.crlUpdateInterval	integer number	600	How often CRLs should be updated, in seconds.
emir.security.trustStore.ocspCacheTtl	integer number	3600	For how long the OCSP responses should be locally cached in seconds (this is a maximum value, responses won't be cached after expiration)
emir.security.trustStore.ocspDiskCache	filesystem path	-	If this property is defined then OCSP responses will be cached on disk in the defined folder.

Property name	Type	Default value / mandatory	Description
emir.security.truststore.ocspLocalResponders	list of properties with a common prefix	-	Optional list of local OCSP responders
emir.security.truststore.ocspMode	[REQUIRE, IF_AVAILABLE, IGNORE]	IF_AVAILABLE	General OCSP ckecking mode. REQUIRE should not be used unless it is guaranteed that for all certificates an OCSP responder is defined.
emir.security.truststore.ocspTimeout	integer number	10000	Timeout for OCSP connections in miliseconds.
emir.security.truststore.revocationOrder	[CRL, OCSP, OCSP_CRL]	OCSP_CRL	Controls overal revocation sources order
emir.security.truststore.revocationUseAll	[true, false]	false	Controls whether all defined revocation sources should be always checked, even if one the first one already confirmed that a checked certificate is not revoked.

4.2.5 Examples

Directory truststore, with a minimal set of options:

```
truststore.type=directory
truststore.directoryLocations=/trust/dir/*.pem
truststore.crlLocations=/trust/dir/*.crl
```

Directory truststore, with complete set of options:

```
emir.security.truststore.type=directory
emir.security.truststore.allowProxy=DENY
emir.security.truststore.updateInterval=1234
emir.security.truststore.directoryLocations=/trust/dir/*.pem http://caserver/ca.pe
emir.security.truststore.directoryEncoding=PEM
emir.security.truststore.directoryConnectionTimeout=100
emir.security.truststore.directoryDiskCachePath=/tmp
emir.security.truststore.crlLocations=/trust/dir/*.crl http://caserver/crl.pem
emir.security.truststore.crlUpdateInterval=400
```

```
emir.security.truststore.crlMode=REQUIRE
emir.security.truststore.crlConnectionTimeout=200
emir.security.truststore.crlDiskCachePath=/tmp
```

Openssl truststore:

```
emir.security.truststore.type=openssl
emir.security.truststore.opensslPath=path/to/truststores/openssl
emir.security.truststore.opensslNsMode=EUGRIDPMA_GLOBUS_REQUIRE
emir.security.truststore.allowProxy=ALLOW
emir.security.truststore.updateInterval=1234
emir.security.truststore.crlMode=IF_VALID
```

Java keystore used as a truststore:

```
emir.security.truststore.type=keystore
emir.security.truststore.keystorePath=path/to/truststores/emir-truststore.jks
emir.security.truststore.keystoreFormat=JKS
emir.security.truststore.keystorePassword=xxxxxx
```

4.3 Configuring the Credentials

EMIR uses private key and a corresponding certificate (called together as a *credential*) to identify clients and servers. The credentials can be provided in several formats. The following table list all possible variants and corresponding parameters.

Property name	Type	Default value / mandatory	Description
`emir.security.credential.format`	[jks, pkcs12, der, pem]	jks	format of the credential
`emir.security.credential.keyAlias`	string	–	keystore alias of the key entry to be used (can be ignored if the keystore contains only one key entry)
`emir.security.credential.keyPassword`	string	–	private key password in keystore (if different from the main credential password)
`emir.security.credential.keyPath`	string	–	location of the private key if stored separately from the main credential

Property name	Type	Default value / mandatory	Description
`emir.security.credential.password`	string	-	password required to load the credential
`emir.security.credential.path`	filesystem path	*mandatory to be set*	credential location

4.3.1 Examples

Credential as a pair of DER files:

```
credential.format=der
credential.password=emi
credential.path=path/to/credentials/cert-1.der
credential.keyPath=path/to/credentials/pk-1.der
```

Credential as a JKS file (type can be autodetected in almost every case):

```
credential.path=path/to/credentials/server1.jks
credential.password=xxxxxx
```

4.4 Authorization

The EMIR offers two alternative options to authorise its' clients.

- Using Access Control List (ACL)
- XACML Policy based authorization

4.4.1 ACL Based Authorization

This is the default mechanism to access control the *Create*,*Update*, and *Delete* operations on EMIR's SER database. The client SERP or child DSR registering SERPs with a parent DSR/GSR get authorised while matching it's distinguished name (DN) against the pre-defined ACL file (CONF/emir.acl). Whereas the file contains a list of DN and role pairs, separated by *::* symbol, see the example below:

the property in the `CONF/emir.config` file

Property name	Type	Default value	Description
`emir.security.access.control.acl`	filesystem path	`CONF/emir.acl`	The location of the ACL file

ACL file contents

```
emailAddress=emiregistry@user.eu,CN=EMIRegistry-Demo-User,OU=JSC,O= ↩
    Forschungszentrum Juelich GmbH,L=Juelich,C=DE :: serviceowner
emailAddress=emiregistry@user.eu-admin,CN=EMIRegistry-Demo-User- ↩
    Admin,OU=JSC,O=Forschungszentrum Juelich GmbH,L=Juelich,C=DE :: ↩
     admin
```

The public key certificate or DN should be sent to the DSR administrator for successfull SER registrations.

Roles: There are only two pre-defined roles within the scope of ACL file:

- the `admin` is a super user who can change any registration, owned by anyone, and
- the `serviceowner` is only allowed to create new or modify his created SERPs.

Important
The ACL based authorisation is only activated when the DSR/GSR is running on SSL/TLS mode

4.5 Policy Based Authorization with XACML

Using XACML 2.0 is an alternative way to authorize clients (User, EMIR-SERP, DSR, or GSR) in a fine grained manner. The administrator should review the policies defined in the `CONF/xacml2Policies/` folder and change them according to her infrastructure needs. However the already defined policies provides a good starting point to the administrators to define/modify the policies.

Important
the XACML policy based authorization will be ignored, if the ACL based authorisation is activated

In order to enable the XACML based authorization: attribute sources and policies must be configured.

4.5.1 Setting Attribute Sources

EMIR only supports file based attribute sources; the client DNs can be included in the attributes file.

Table 1: File Attribute Source Settings

Property name	Type	Default value	Description
`emir.security.attributes.order`	string	`FILE`	This property is a space separated list of attribute source names, which are then configured in detail below. The named attribute sources are queried in the given order.
`emir.security.attributes.FILE.class`	string	`eu.emi.emir.aip.FileAttributeSource`	configuration of the *FILE* attribute source
`emir.security.attributes.FILE.matching`	[strict, regexp]	`strict`	Specifies the matching or client DNs
`emir.security.attributes.FILE.path`	filesystem path	`CONF/users/test.xml` or `CONF/users/test-regexp.xml`	The path to the file containing subjects' DNs. The file suffixed with `strict` use strict checking of DNs, whereas the file suffixed with `regexp` contain entries using regualr expressions

4.5.2 Setting XACML Policies

Table 2: Policy settings

Property name	Type	Default value	Description
`emir.security.accesscontrol.pdp.config`	filesystem path	`CONF/xacml2.conf`	The path to the XACML2 configuration, containing the rules of executing the policies
`emir.security.accesscontrol.pdp`	string	`eu.unicore.uas.pdp.local.LocalHerasafPDP`	The name of the PDP class to endorse, for the xacml2 policies execution

The `CONF/xacml2.config` file contains raw xacml polices, enable EMIR (DSR or GSR) administrators to write their own rules.

4.6 MongoDB Database Configuration

The EMIR uses MongoDB to store and index the SER collections. It must be configured and running before deploying any EMIR (DSR or GSR) server.

Property name	Type	Default value	Description
Connection Settings			
`mongodb.hostName`	string	`localhost`	Fully qualified host name of the machine on which MongoDB is setup
`mongodb.port`	Integer	`27017`	The port number
Database Settings			
`mongodb.dbName`	string	`emiregistry`	The name of the database to store the SERP records
`mongodb.colName`	string	`services`	The name of the collection (of the database) in which the records will be stored
Login Settings			
`mongodb.userName`	string	–	The username to access the MongoDB database
`mongodb.password`	string	–	The password to access the MongoDB database

For high loads, especially at the GSR level, it is recommended to setup MongoDB replication for enhanced scalability and performance.

4.7 Building EMIR Network

EMIR allows building a network of registries participating in a Grid infrastructure or federation. The network can be of type hierarchical or Peer-to-Peer (P2P). In an hierarchical network, the SER collections are propagated from leaf DSR node to the top level root node, called GSR. Each DSR has only one parent, either DSR or GSR to which it pushes it's SER collections. At the root level the P2P network of GSR is formed to replicate the SER collections among multiple GSRs by referring a pre-configured `Global list`. The global list contains a listing of URLs of all the GSRs, each of which should be able to access the URLs.

4.7.1 How to Setup DSR?

In order to build hierarchy of DSRs must be able to propagate the SER collections to any **single** parent DSR or a GSR.

Table 3: Parent DSR Settings

Property name	Type	Default value	Description
EMIR's DSR settings			
`emir.parentAddress`	string	`http://localhost:9127`	The address/URL (http or https) of the EMIR DSR server to which it propagates its SER collection

Important
Add DN of child DSR into the parent DSRs CONF/emir.acl or CONF/users/testUd-(strict | regsexp).xml

4.7.2 How to Setup GSR?

The root level GSR has two primary functions:

- aggregation of children DSR SER collections
- replicating the SER collections among other GSRs (visible of *Global List*)

Table 4: GSR Settings

Property name	Type	Default value	Description
EMIR's DSR settings			
`emir.parentAddress`	string	`http://localhost:9127`	The address/URL (http or https) of the EMIR server to which it propagates its information
General GSR Settings			

Table 4: (continued)

Property name	Type	Default value	Description
`emir.global.enable`	boolean	`false`	If set to true, indicating the registry node is global. It will then replicate the state among peer global registries (GSRs), the *emir.parentAddress* property will be ignored (if enabled), as the root registry should not contain any parent.
`emir.global.sparsity`	Unsigned Integer	`2`	It determines the number of neighbors as a function of the actual number of member nodes of the network.
`emir.global.retry`	Unsigned Integer	`5`	It specifies a number of attempts if communication to another GSR is failed.
`emir.global.etValid`	Unsigned Integer	`12`	Specifies period in hours for checking the entries in the soft state database and strip the expired entries (but still keeps them).
`emir.global.softStateDelay`	Integer	`2`	Extend the expiration time with this time delay in hours.
`emir.global.etRemove`	Integer	`24`	Specifies period in hours for checking the entries in the soft state database and remove the expired entries.
Global List Settings			
`emir.global.providerURL`	URL or filesystem path	–	Link to the document listing GSR URLs. The URL(s) is/are important for building the GSR's P2P network at the global level.

Important
Add DN of child DSR into the GSRs CONF/emir.acl or CONF/users/testUd-(strict | regsexp).xml

4.8 Service Endpoint Record (SER) Management

4.8.1 Setting Service Endpoint Records (SER) Lifetime

In EMIR, every SER has associated lifetime or (Time-To-Live) TTL. The settings can be defined in DSR or GSR to restrict the maximum assignable lifetime and assign default lifetime if missing from the registration.

Table 5: SER TTL Settings

Property name	Type	Default value	Description
`emir.record.expiryMax`	Unsigned Integer (in days)	–	Maximum assignable lifetime for the SERs containing the Service_ExpireOn property, defined in days, minimum value: 1.
`emir.record.expiryDefault`	Unsigned Integer	–	The default lifetime will be set from the given property if the incoming registration is without the Service_ExpireOn attribute.

4.8.2 Filtering Service Endpoint Records (SER)

EMIR offers a way to block

- SERs from being registered via DSR or EMIR-SERP
- SERs from being propagated to it's parent DSR or GSR

Table 6: SER Filter Settings

Property name	Type	Default value	Description
`emir.record.blockListFiltering.inputfile`	filesystem path	`CONF/inputfilter.ls`	The file containing list of SER IDs, matching services will be blocked from registration to it's index
`emir.record.blockListFiltering.outputfile`	filesystem path	`CONF/outputfilter.ls`	The file containing list of SER IDs, matching services will be blocked from propagation to it's parent DSR

4.9 Logging Configuration

The EMIR server uses log4j to provide log facilities to record all but some of the server activities. In order to change the logging configuration, CONF/log4j.properties should be reviewed by the administrator.

Property name	Type	Default value	Description
`log4j.configuration`	filesystem path	`STARTUP_SCRIPT`	The log4j configuration

4.10 Advanced HTTP Server Settings

EMIR uses Eclipse's Jetty server to host REST Web services. Following table lists the important properties.

Important
Do not set **emir.jetty.requireClientAuthn** and **emir.jetty.wantClientAuthn** in CONF/emir.config file, as they are automatically set by the EMIR server on start-up.

Property name	Type	Default value / mandatory	Description
`emir.jetty.disableCipherSuites`	string	*empty string*	Space separated list of SSL cipher suites to be disabled.

Property name	Type	Default value / mandatory	Description
emir.jetty.fastRandom	[true, false]	false	Use insecure, but fast pseudo random generator to generate session ids instead of slow and secure generator for SSL sockets. Useful for testing.
emir.jetty.gzip.enable	[true, false]	false	Controls whether to enable compression of HTTP responses.
emir.jetty.gzip.minGzipSize	integer number	100000	Specifies the minimal size of message that should be compressed.
emir.jetty.highLoadConnections	integer > 1	200	If the number of connections exceeds this amount, then connector is put into a special *low on resources* state. Existing connections will be closed faster. Note that this value is honored only for NIO connectors. Legacy connectors go into low resources mode when no more threads are available.
emir.jetty.lowResourceMaxIdleTime	integer > 1	10000	In low resource conditions, time (in ms.) before an idle connection will time out.
emir.jetty.maxIdleTime	integer > 1	200000	Time (in ms.) before an idle connection will time out. It should be large enough not to expire connections with slow clients, values below 30s are getting quite risky.
emir.jetty.maxThreads	integer > 1	255	Maximum number of threads to have in the Jetty thread pool for connections serving.
emir.jetty.minThreads	integer > 1	1	Minimum number of threads to have in the Jetty thread pool for connections serving.

Property name	Type	Default value / mandatory	Description
`emir.jetty.requireClientAuthn`	[true, false]	`true`	Controls whether the SSL socket requires client-side authentication.
`emir.jetty.soLingerTime`	integer number	`-1`	Socket linger time.
`emir.jetty.useNIO`	[true, false]	`false`	Controls whether the NIO connector be used. NIO is best suited under high-load, when lots of connections exist that are idle for long periods.
`emir.jetty.wantClientAuth`	[true, false]	`true`	Controls whether the SSL socket accepts client-side authentication.

5 EMIR Service Endpoint Record Publisher (EMIR-SERP) Client

5.1 About the EMIRD

The UMD services need to be registered into the EMI Registry service infrastructure to be discoverable for the clients. Most of the services or even the containers executing them provide a way to do this but not all of them. For those that are unable to register themselves automatically and periodically the EMIRD is available.

The EMIRD is a daemon like (background) service that can be executed next to these services (preferably on the same machine) and able to perform the automatic and periodical registration and update against the configured EMI Registry service on behalf of the service itself. This client uses exactly the same, standard RESTful API as the other clients do.

Most of the parameters of these registrations and updates can be configured. For the details see the Configuration section!

After the successful registration until the termination of the daemon, the EMIRD client do the periodical updates then finally, when the execution of the daemon is over, it attempts to delete the service entries from the remote database.

The service entries can be simple or advanced ones.

The simple service entries contain only the mandatory and easily configurable attributes that are the following: Service_Name, Service_Type, Service_Endpoint_URL, Service_Endpoint_InterfaceName. Here, the single mandatory element is the Service_Endpoint_URL.

The advanced entries can contains any kind of key value pairs that are accepted by the EMI Regisrty services and can be configured in the form of whole, formatted **json** documents.

5.2 Installation

The installation of the EMIRD client is trivial. The only thing to do is to install the `emir-daemon` package from the EMI repository by executing:

```
yum instlal emir-daemon
```

5.3 Configuration

The configuration of EMIRD can be performed by editing its configuration file or files. The configuration can be found basically in one file that default location is `/etc/emi/emird/emird.ini`.

This file contains every configuration options that can be the EMIRD daemon control by, like *service url*, *logging verbosity*, *credential location*, etc.

The advanced service entries to be propagated can be described in separated configuration files preferably also under this directory and use to have `.json` extension.

The main configuration file has INI format. The `emir` section contains the daemon scoped options while the others are to describe the different service entries to be registered. In these cases the exact name is indifferent, they just have to differ from eachother and must avoid the `emir` name as well.

5.3.1 Configuration options

Note: The option names are case-insensitives.

url

`Location:` `emir` section

`Default value:` *No default value*

`Mandatory:` Yes

`Description:`

URL of the EMIR service to connect in a protocol://domain:port format.

The protocol part is not mandatory if https (default) The port part is not mandatory if 54321 (default) The domain part is mandatory

Examples: url = emiregistry2.grid.niif.hu url = https://emiregistry2.grid.niif.hu url = https://emiregistry2.grid.niif.hu:54321

period

`Location:` `emir` section

`Default value:` *No default value*

`Mandatory:` Yes

`Description:`

The period of the registration/update messages. Its value is given in minutes.

validity

`Location:` `emir` section

`Default value:` *No default value*

`Mandatory:` Yes

`Description:`

The validity of the registration entries. Its value is given in minutes.

cert

`Location:` `emir` section

`Default value:` /etc/grid-security/hostcert.pem

`Mandatory:` No

`Description:`

User certificate file location in PEM format. Only used and checked if the protocol in the url option is *https*.

key

`Location:` `emir` section

`Default value:` /etc/grid-security/hostkey.pem

`Mandatory:` No

`Description:`

User key file location in PEM format. Only used and checked if the protocol in the url option is *https*.

cadir

`Location:` `emir` section

`Default value:` /etc/grid-security/certificates

`Mandatory:` No

`Description:`

A path pointing to the store where the PEM certificate of the trusted Certificate Authorities can be found. Only used and checked if the protocol in the url option is *https*.

verbosity

`Location:` `emir` section

`Default value:` error

`Mandatory:` No

`Description:`

Logging verbosity. The parameter is optional. If missing or an invalid value is given, the default value will be used. The logs are written into the log file that can be found in the */var/log/emi/emird* directory by default.

Service_Endpoint_URL

`Location:` simple service entry section

`Default value:` *No default value*

`Mandatory:` Yes

`Description:`

The Service Endpoint URL to be propagated. If this option is missing an error message will be raised.

Service_Name

`Location:` simple service entry section

`Default value:` *No default value*

`Mandatory:` No

`Description:`

The Service Name to be propagated. If this option is missing then the service entry will contains no such component.

Service_Type

`Location:` simple service entry section

`Default value:` *No default value*

`Mandatory:` No

`Description:`

The Service Type to be propagated. If this option is missing then the service entry will contains no such component.

Service_Endpoint_InterfaceName

`Location:` simple service entry section

`Default value:` *No default value*

`Mandatory:` No

`Description:`

The Service Endpoint Interface Name to be propagated. If this option is missing then the service entry will contains no such component.

json_file_location

`Location:` advanced service entry section

`Default value:` *No default value*

`Mandatory:` Yes

`Description:`

The service entry can be also defined in a single external json formatted file per service. Any allowed json attributes are allowed in this way. The location of this file must be defined in this ini variable. The recommended place for these files is under the `/etc/emi/emird/` directory and naming them after the name of the given service with `.json` extension.

6 REST API

The EMI Registry allows Services to register/publish their capabilities while the Service Consumers are able to find the deployed services.

This section contains the description of the REST-ful interface, that allows the management of the service information (or entries) by exposing the individual URIs. The normative description of the API cab also be defined as Web Application Description Language (WADL) document WADL Section 8.

6.1 Register new Services

`HTTP Method` : POST

`URI` : /serviceadmin

`Content Type` : application/json

`Security Implications` : Requires authenticated "and" authorized user access to perform this operation

6.1.1 Request

The message body contain a JSON Array containing the JSON objects (see below), each of which would be a service entry in the EMI registry.

Service description is defined as a Section 7 document.

Important
The only mandatory attribute is **Service_Endpoint_URL**, which should be unique

6.1.2 Response

The response contains similar array of JSON Objects as it was in sent request, confirming the successful update.

`Status Code` : OK / 200

6.2 Updating the Service information

`HTTP Method` : PUT

`URI` : /serviceadmin

`Content Type` : application/json

`Security Implications` : Requires an authenticated "and" authorized user access to perform this operation

6.2.1 Request

The request body contain a similar JSON array object as defined POST method that contains the description of the Services to be updated. The Service Entries identified by the *Service_Endpoint_URL* key in the individual JSON objects will be updated respectively.

6.2.2 Response

The response contains similar array of JSON Objects as it was in sent request, confirming the successful update.

`Status Code` : OK / 200

6.3 Delete existing Services

`HTTP Method` : DELETE

`URI` : /serviceadmin

`Security Implications` : Requires an authenticated "and" authorized user access to perform this operation

6.3.1 Request

The Service Entry matching the Endpoint ID will be deleted from the registry only if the client executing the action has authorised access and the method is allowed by the security plugins.

`Query Parameters` : Service_Endpoint_ID= <*Service unique Endpoint ID*>

Example : /serviceadmin?Service_Endpoint_ID=urn:endpoint:emi1

6.3.2 Response

`Status Code` : OK / 200

6.4 Querying the EMIR

`HTTP Method` : GET

`URI` : /services

`Content Type` : application/json

6.4.1 Request

The request contains the key-value pairs separated by ampersand `&`

`Query Parameters` : AttributeName=<Attribute_Value>&AttributeName=<Attribute_Value>&...

Example : /services?Service_Type=eu.emi.es&Service_Endpoint_HealthState=ok

The additional parameters can also be added to restrict and/or paginate the result

`Additional Query Parameters:`

```
skip=Integer value
```

skip returns the result skipping the given number of entries

```
limit=Integer value
```

limit defines the maximum number of result containing the service entries

Response+Additional Query Parameters+ :

```
skip=Integer value
```

skip returns the result skipping the given number of entries

```
limit=Integer value
```

limit defines the maximum number of result containing the service entries

The response contains an array of service entries packed in a JSON array object

`Status Code` : OK / 200

6.5 Rich Querying in EMIR

`HTTP Method` : GET

`URI` : /services

`Content Type` : application/json

6.5.1 Request

The request contains the JSON document including with support for defining advanced clauses, the http://www.mongodb.org/display/DOCS/Advanced+Queries, MongoDB Advanced Queries[MongoDB JSON Query Language] describes the various types of queries

Additional keys (skip, limit) can also be added to paginate the returning results.

6.5.2 Response

The response contains the array of service entries packed in a JSON array object

`Status Code` : OK / 200

6.6 Querying the EMIR for GLUE 2.0 XML Documents

`HTTP Method` : GET

`URI` : /services

`Content Type` : application/xml

6.6.1 Request

The request contains the key-value pairs separated by ampersand `&`

`Query Parameters` : AttributeName=<Attribute_Value>&AttributeName=<Attribute_Value>&...

Example : /services?Service_Type=eu.emi.es&Service_Endpoint_HealthState=ok

The additional parameters can also be added to restrict and/or paginate the result

`Additional Query Parameters` :

```
skip=Integer value
```

skip returns the result skipping the given number of entries

```
limit=Integer value
```

limit defines the maximum number of result containing the service entries

6.6.2 Response

The response contains an XML document containing service entries in GLUE 2.0 format

`Status Code` : OK / 200

6.7 Rich Querying the EMIR for GLUE 2.0 XML Documents

The request and response interface is same as defined above, however the content type must be defined as **application/xml** instead.

6.8 Viewing the Service information template

This To view the GLUE 2.0's JSON flavored service model.

`HTTP Method` : GET

`URI` : /model

`Content Type` : application/json

6.8.1 Request

N/A

6.8.2 Response

JSON document, as described in the `/serviceadmin` POST method

`Status Code` : OK / 200

6.9 Monitoring the Registry

Allows registry users to view the registry status

`HTTP Method` : GET

`URI` : /ping

6.9.1 Request

N/A

6.9.2 Response

`Status Code` : OK / 200

7 Appendix I

The service record JSON template of EMIR interface.

```
[       //Example Service Endpoints Records (belonging to the same  ↩
    service)
        {
                "Service_ID":"s1",
                "Service_Name":"ComputingService",
                "Service_CreationTime":{"$date":"2011-07-21T11 ↩
                    :47:24Z"},
                "Service_Type":"job-management",
                "Service_Contact": [{"ContactType":"sysadmin", " ↩
                    Detail":"http://contactlink"},{"ContactType":" ↩
                    developer", "Detail":"http://contactlink"}],
                "Service_Endpoint_ID":"se1", //this should be  ↩
                    unique
                "Service_Endpoint_URL":"http://1",
```

```
                "Service_Endpoint_Capability":["capability1"," ↩
                    capability2"],
                "Service_Endpoint_Technology":"technology",
                "Service_Endpoint_InterfaceName":"interface",
                "Service_Endpoint_InterfaceVersion":["version1"," ↩
                    version2"],
                "Service_Endpoint_InterfaceExtension":["extension1 ↩
                    ","extension2"],
                "Service_Endpoint_WSDL":"http//1.wsdl",
                "Service_Endpoint_SupportedProfile":["profile1"," ↩
                    profile2"],
                "Service_Endpoint_Semantics":["semantic1"," ↩
                    semantic2"],
                "Service_Endpoint_HealthState":"ok",
                "Service_Endpoint_HealthStateInfo":"state info",
                "Service_Endpoint_ServingState":"production",
                "Service_Endpoint_StartTime":{"$date":"2011-07-21 ↩
                    T11:47:24Z"},
                "Service_Endpoint_DowntimeAnnounce":{"$date ↩
                    ":"2011-07-21T11:47:24Z"},
                "Service_Endpoint_DowntimeStart":{"$date ↩
                    ":"2011-07-21T11:47:24Z"},
                "Service_Endpoint_DowntimeEnd":{"$date":"2011-07-21 ↩
                    T11:47:24Z"},
                "Service_Endpoint_QualityLevel":"production",
                "Service_Location_Address":"A Street 1",
                "Service_Location_Place":"Bonn",
                "Service_Location_Country":"Germany",
                "Service_Location_PostCode":"53119",
                "Service_Location_Latitude":53.3,
                "Service_Location_Longitude":4,
                "Service_ExpireOn":{"$date":"2020-07-21T11:47:24Z"}

        },
        {
                "Service_ID":"s1",
                "Service_Name":"ComputingService",
                "Service_CreationTime":{"$date":"2011-07-21T11 ↩
                    :47:24Z"},
                "Service_Type":"job-management",
                "Service_Contact": [{"ContactType":"sysadmin", " ↩
                    Detail":"http://contactlink"},{"ContactType":" ↩
                    developer", "Detail":"http://contactlink"}],
                "Service_Endpoint_ID":"se2", //this should be  ↩
                    unique
                "Service_Endpoint_URL":"http://1",
                "Service_Endpoint_Capability":["capability1"," ↩
                    capability2"],
                "Service_Endpoint_Technology":"technology",
                "Service_Endpoint_InterfaceName":"interface",
```

```
                "Service_Endpoint_InterfaceVersion":["version1"," ↩
                    version2"],
                "Service_Endpoint_InterfaceExtension":["extension1 ↩
                    ","extension2"],
                "Service_Endpoint_WSDL":"http//1.wsdl",
                "Service_Endpoint_SupportedProfile":["profile1"," ↩
                    profile2"],
                "Service_Endpoint_Semantics":["semantic1"," ↩
                    semantic2"],
                "Service_Endpoint_HealthState":"ok",
                "Service_Endpoint_HealthStateInfo":"state info",
                "Service_Endpoint_ServingState":"production",
                "Service_Endpoint_StartTime":{"$date":"2011-07-21 ↩
                    T11:47:24Z"},
                "Service_Endpoint_DowntimeAnnounce":{"$date ↩
                    ":"2011-07-21T11:47:24Z"},
                "Service_Endpoint_DowntimeStart":{"$date ↩
                    ":"2011-07-21T11:47:24Z"},
                "Service_Endpoint_DowntimeEnd":{"$date":"2011-07-21 ↩
                    T11:47:24Z"},
                "Service_Endpoint_QualityLevel":"production",
                "Service_Location_Address":"A Street 1",
                "Service_Location_Place":"Berlin",
                "Service_Location_Country":"Germany",
                "Service_Location_PostCode":"53011",
                "Service_Location_Latitude":53.5,
                "Service_Location_Longitude":4,
                "Service_ExpireOn":{"$date":"2020-07-21T11:47:24Z"}
        }

]
```

8 Appendix II

The EMIR WADL document to define the REST-ful API

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<application xmlns="http://wadl.dev.java.net/2009/02">
    <doc xmlns:jersey="http://jersey.java.net/" jersey:generatedBy ↩
        ="Jersey: 1.9.1 09/14/2011 02:05 PM"/>
    <grammars/>
    <resources base="https://localhost:54321/">
        <resource path="/children">
            <method id="childDSRs" name="GET">
                <response>
                    <representation mediaType="*/*"/>
                </response>
```

```
            </method>
            <method id="checkin" name="POST">
                <response>
                    <representation mediaType="*/*"/>
                </response>
            </method>
        </resource>
        <resource path="/neighbors">
            <method id="childDSRs" name="GET">
                <response>
                    <representation mediaType="*/*"/>
                </response>
            </method>
        </resource>
        <resource path="/parent">
            <method id="childDSRs" name="GET">
                <response>
                    <representation mediaType="*/*"/>
                </response>
            </method>
        </resource>
        <resource path="/serviceadmin">
            <method id="getServicebyUrl" name="GET">
                <response>
                    <representation mediaType="application/json"/>
                </response>
            </method>
            <method id="registerServices" name="POST">
                <request>
                    <representation mediaType="application/json"/>
                </request>
                <response>
                    <representation mediaType="application/json"/>
                </response>
            </method>
            <method id="updateServices" name="PUT">
                <request>
                    <representation mediaType="application/json"/>
                </request>
                <response>
                    <representation mediaType="application/json"/>
                </response>
            </method>
            <method id="deleteService" name="DELETE">
                <response>
                    <representation mediaType="*/*"/>
                </response>
            </method>
        </resource>
        <resource path="/services">
```

```
            <method id="queryWithParams" name="GET">
                <response>
                    <representation mediaType="application/json"/>
                </response>
            </method>
            <method id="queryWithJSON" name="POST">
                <request>
                    <representation mediaType="application/json"/>
                </request>
                <response>
                    <representation mediaType="application/json"/>
                </response>
            </method>
            <method id="queryXMLWithJSON" name="POST">
                <request>
                    <representation mediaType="application/json"/>
                </request>
                <response>
                    <representation mediaType="application/xml"/>
                </response>
            </method>
            <method id="queryXMLWithParams" name="GET">
                <response>
                    <representation mediaType="application/xml"/>
                    <representation mediaType="text/xml"/>
                </response>
            </method>
            <resource path="/urls">
                <method id="getServiceEndPoints" name="GET">
                    <response>
                        <representation mediaType="application/json ↩
                            "/>
                    </response>
                </method>
            </resource>
            <resource path="/types">
                <method id="getServiceTypes" name="GET">
                    <response>
                        <representation mediaType="application/json ↩
                            "/>
                    </response>
                </method>
            </resource>
            <resource path="/query.xml">
                <method id="queryXml" name="GET">
                    <response>
                        <representation mediaType="application/xml ↩
                            "/>
                        <representation mediaType="text/xml"/>
                    </response>
```

```
                </method>
            </resource>
            <resource path="/pagedquery">
                <method id="pagedQuery" name="GET">
                    <response>
                        <representation mediaType="*/*"/>
                    </response>
                </method>
            </resource>
        </resource>
        <resource path="/model">
            <method id="getModel" name="GET">
                <response>
                    <representation mediaType="text/html"/>
                    <representation mediaType="application/json"/>
                </response>
            </method>
        </resource>
        <resource path="/ping">
            <method id="ping" name="GET">
                <response>
                    <representation mediaType="application/json"/>
                    <representation mediaType="text/plain"/>
                </response>
            </method>
        </resource>
    </resources>
</application>
```